# Security model for hybrid token-based networking models

Rudy Borgstede ( Rudy.Borgstede@gmail.com )
University of Amsterdam

Version 1.0

February 4, 2008

# Abstract

**This project report** is written as a contribution to the GAAA Toolkit development project to research on a new security model for Token Based Networking using Identity Based Cryptography.

**The report gives an overview of the Identity-Based Cryptography (IBC) model,** which is considered as an alternative to widely used Public Key Infrastructure (PKI). With the use of IBC there is no need anymore for a Certificate Authority, as with Public Key Infrastructure, which solves some key distribution and scalability issues. This gives the client the possibility to use cryptography without even being connected to the internet. The report investigates the possibility to use IBC based token key distribution comparing to currently used shared secret model and alternative solution based on PKI. Two existing implementations of IBC of Voltage called Identity-Based Encryption and the National University of Ireland in Maynooth called Eyebee are compared and evaluated. The Eyebee solution is also tested by a short experiment.

# Contents

# Chapter 1

# Introduction

Before the report starts about Identity-Based Cryptography, first the project background is described.

This project is conducted in the framework of the System and Network Engineering[6] master education of the University of Amsterdam[7] for the course Research Project 1. The main goal of this course is getting a taste of the real world work. Besides the work on the project and a presentation is the product of this course described by the master as a consultancy like report with an analytic vision[1].

This research is done as a contribution to the GAAA Toolkit project[2] to develop IBC based security model for token keys[19] distribution by Token Validation Service (TVS) what components of the Token Based Networking (TBN) infrastructure, referred hereafter as TBS-TVS project. The current TVS implementation In the form of aaauthreach-tvs package can work with binary and XML[10] based token. The current TVS Implementation uses shared secret security key distribution model and HMAC-SHA1 transformation for token generation. Shared secret key distribution model has known scalability and manageability problems for an open network environment. So could it be replaced?

As this report is focused on *Identity-Based Cryptography*, it concentrates on why it is a better solution and proving why it does a better job for the TBS-TVS project then other cryptography models like *Public Key Infrastructure[4]* and *Kerberos[16]*.

The report has the following chapters:

- Identity-Based Cryptography Basics. How the Identity-Based Cryptography model works and what advantages or disadvantages it has?
- Implementation. This chapter evaluates two opensource Identity-Based Cryptography solutions and Java[3] classes which are written to test the Eyebee solution.
- Experiment and Evaluation. This is a short experiment to research how usefull the Eyebee[8] solution is for the TBS-TVS project.
- Conclusion. This chapter gives an opinion about the usefullnes of Identity-Based Cryptography for the TBS-TVS project.
- Further Research. This chapter contains subjects related to the Identity-Based Cryptography model which should be further researched.

- Appendix. This last chapter has several documents which are related to the report like the bibliography, the planning, a personal log and the original approved project description.

## 1.1 Use Case

In this section two examples of the two basic implementations of Identity-Based Cryptography model are explained.

### 1.1.1 Server-Based Identity

Identity-Based Cryptography systems which are based on the server identity are systems where a token is encrypted and decrypted by the identity or so called address of the server e.g. www.google.nl or Force10. Think of a network described by the "Network Description Language[5]" see figure 1.1, which is a XML[10] based markup language to describe network resources and their location, which can make the network environment known to any identity in the network. This makes it possible to encrypt a token for the server which contains the right resource. So the token can be securely passed through an open network and can only be decrypted by the right destination server.

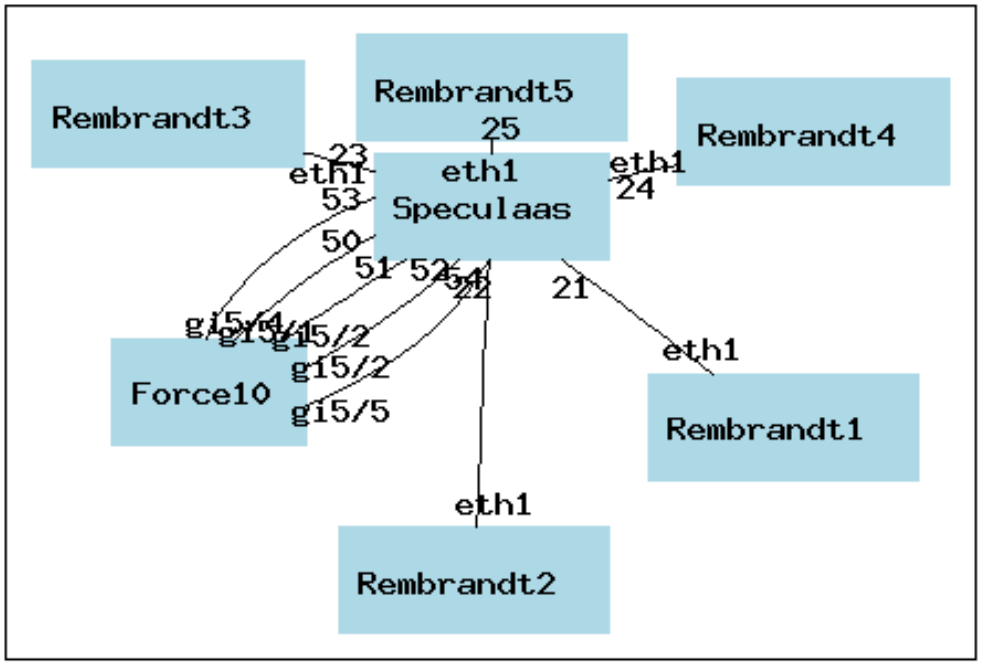


Figure 1.1: The example network infra structure from the NDL website[5]

### 1.1.2 User-Based or Service-Based Identity

Identity-Based Cryptography systems which are based on the user or service identity are systems where a token key is encrypted and decrypted by the address, identifier or name

of the user or service e.g. Rudy.Borgstede@gmail.com, Rudy Borgstede, Security Camera Garage or Studentnumber: 123456789. Which means that the encrypted token doesn't have a complete address where it can be delivered for decryption but rather needs an external defined address that points to the location of the user or service (identity) which decrypts the token. This means that the location of the user or service must be known, which is not always possible because of the architecture of the network. A good example of an User-Based or a Service-Based Identity system, where Identity-Based Cryptography can be used, is an instant messenger network like "ICQ" or "MSN". In such systems the location of each identity is known indirect and is referred to by a unique number or mail address, which *is* a User-Based Identity. The result of such a system is that the identity could have a dynamic location in the network and can an identity e.g. a secure mail server be replicated to multiply locations in the network.

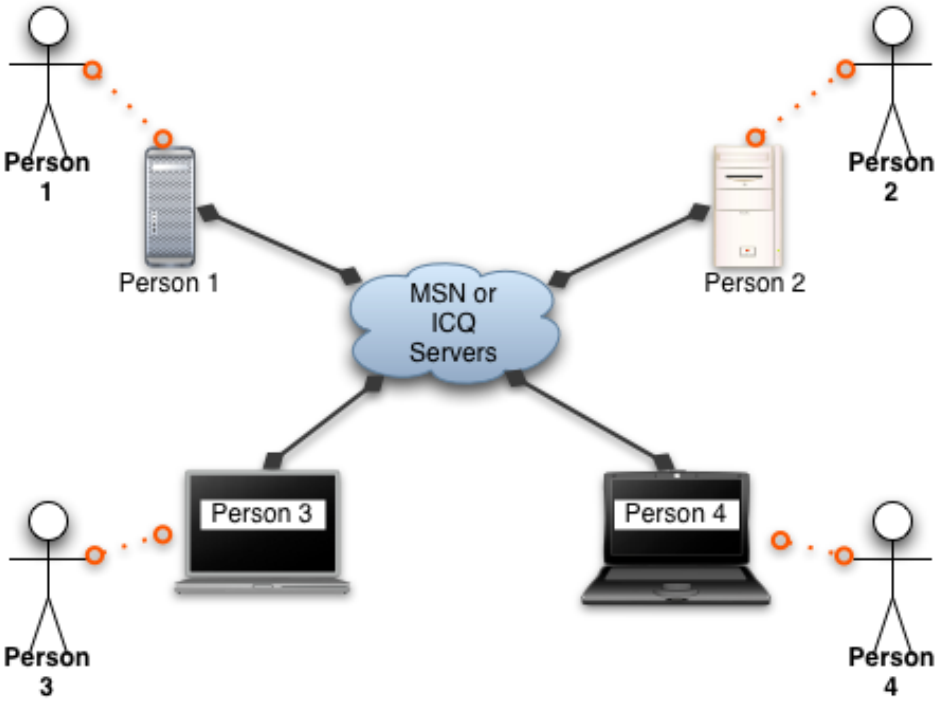


Figure 1.2: An abstract view of an instant messenger network. A network with many different systems which contains similar like user-based identities.

# Chapter 2

# Identity-Based Cryptography Basics

In this chapter the Identity-Based Cryptography model is further explained and compared against the other cryptography models. Also the security of the Identity-Based Cryptography model is evaluated.

## 2.1 How does the RSA a-symmetric key cryptography model work?

In this section the basic idea about RSA a-symmetric key cryptography[18] is explained.
The cryptography model exists out of three components generated by the RSA algorithm[18]:

- **The private master key** is a big generated number, which should remain a secret. Before the private master key could be calculated first a big number (n) is generated by two big prime numbers (p and q).

  $$n = p \times q \tag{2.1}$$

  The big number n is used by the modulo to encrypt and decrypt e.g. a token. Next the totient[13] is calculated of the two base prime numbers (p and q). A totient is the number of co-prime numbers in n.

  $$\phi(n) = (p-1) \times (q-1) \tag{2.2}$$

  Now the private master key is $\phi(n)$ the torient.

- **The public certificate or key** is based on the identity of the destination like a server, service or user identity. This generated public certificate or key can be seen as the identity of the destination. The difference between a public certificate and key is that a public certificate has a public key. The public key is used for encrypting e.g. a token. A public certificate is a collection of information about an identity like when the public certificate becomes valid or invalid but also the name and address of the identity. A public certificate is used by the Public Key Infrastructure model and only a public key by the Identity-Based Cryptography model. The extra information that comes with a public certificate is not needed for Identity-Based Cryptography because the client makes the public key it self. This means that it isn't usefull to generate a public certificate with information that the client obviously already knows, the public key is

used to confirm the identity of the destination. In the formula of the private master key could n be seen as the public key, which e.g. with Identity-Based Cryptography is being combined with the identity of the destination to form the final public key.

- **The private key** is a number generated from the private master key. This key is generated so that the client can with the public certificate or key encrypt e.g. a token but can only decrypt it with the private master key. The private key is a number between one and the private master key (which is a big number). Also the chosen number must be a co-prime with the private master key:

$$1 < privatekey < \phi(n) \tag{2.3}$$

After these components are initialised e.g. the token is encrypted:

$$encryptedtoken = token^{privatekey} \bmod publickey \tag{2.4}$$

To decrypt e.g. the token:

$$calculatedkeytodecrypt \times privatekey \equiv 1 \bmod privatemasterkey \tag{2.5}$$

$$token = encryptedtoken^{calculatedprivatekeytodecrypt} \bmod publickey \tag{2.6}$$

As example the Public Key Infrastructure model is explained with these components. First the server has to prepare the Public Key Infrastructure model:

1. Generate the server private master key.
2. Generate a public certificate.
3. Let the public certificate be signed by a trusted Certificate Authority, this is a third party authority which is highly trusted by the clients of the server, to ensure integrity and authenticity of the certificate. This is achieved because the Certificate Authority can also validate, a previous signed public certificate.
4. Make the public certificate public available e.g. by an apache webserver with SSL.

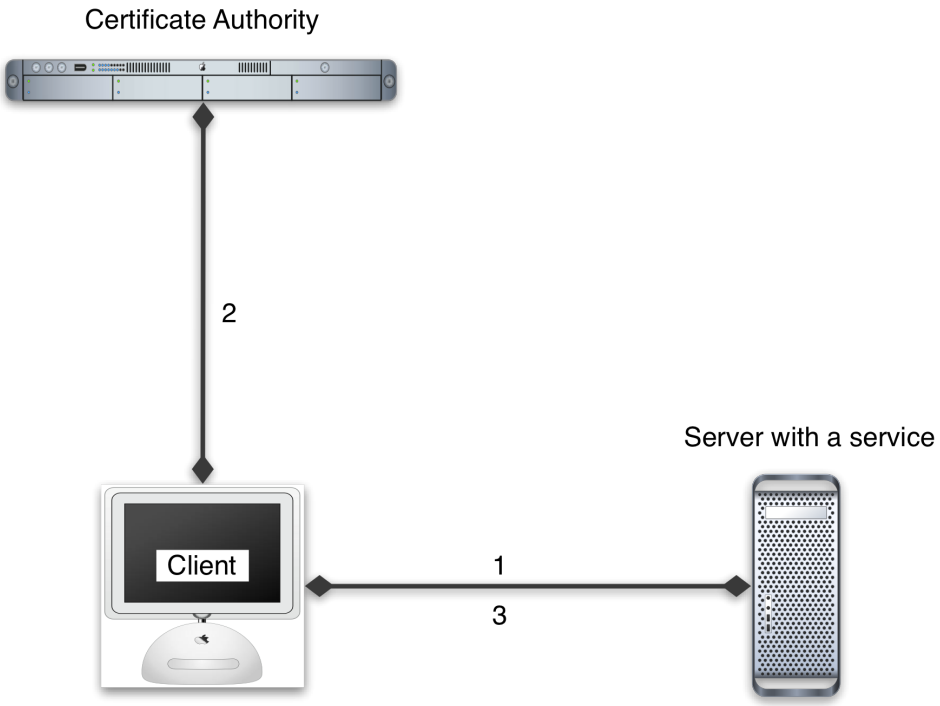


Figure 2.1: An abstract representation of the Public Key Infrastructure.[4]

Now the Public Key Infrastructure model is setup, a client can connect with the following procedure (The enumerated numbers are relative to the numbers from the figure 2.1):

1. The client retrieves the public certificate from the server.
2. The client validates the public certificate by contacting the Certificate Authority.
3. The client negotiates with the server for a private key by using the Diffie-Hellman(-Merkle) algorihm and encrypt and signs e.g. the token with the public certificate and the client private key. Then the e.g. token is send tot the server which decrypt the token with the private key, the public key and the private master key.

The RSA a-symmetric key cryptography model ensures four important security properties[4], if the private master keys aren't compromised:

- **Integrity** means that the e.g. a token isn't changed while transported over the network. The integrity can be assured by signing the encrypted e.g. token. This is done by creating a hash of e.g. the token by using a hashing algorithm[14] like MD5 or SHA-1. Only the client and the server can (re)calculate the same hash because the hash is encrypted so only the server can decrypt and validate the hash. If the integrity is comprimised the recalculation of the hash on the destination server doesn't produce the same answer as the client generated.
- **Confidenciality** means that nobody except the reciever can read the original e.g. token. This is done by the unique a-symmetric key property of the RSA algorithm.

With the use of a public certificate or key and a private key the client can encrypt e.g. the token for the server but can't decrypt it with the same public certificate or key and the private key. For decrypting the private master key is needed. This garantees the confidenciality because only the right destination server has the right private master key. Only with the Identity-Based Cryptography model the client can also decrypt his own message because the client and the server share the same private master key. This still gives the sender confidenciality because this only lets an identity decrypt the data if he already know the data or know the private master key.

- **Authenticity** means that the identity of the server, user or service can be garanteed. This can be garanteed if the public certificate of the server is fully trusted, because the public certificates exactly describes the server and the identity of the server but also gives the client the public key, which the client needs to encrypts his data. This means with the Public Key Infrastructure model that the public certificate should be signed to prove the information about an identity is true. If the public certificate is self-signed then it should only be used for testing because this means that the server says "just trust me for me". This is a problem because the second "me" could be everyone also a hacker who sends a fake public certificate to steal data. To really proof the authenticity of the server a third party authority could validate the authenticity of the public certificate. The means that the server says "trust me because e.g. Verisign says the public certificate is valid". This raises ofcourse the question how trustworthy the Certificate Authority is? If the public certificate is valid then the only one who can decrypt the encrypted e.g. token should be the server which identity has been described by the public certificate. With the Identity-Based Cryptography model a public key is created by the client it self so the authenticity of the destination identity is proven instantly because the client trusts him self to generate the right public key, because with Identity-Based Cryptography the client should already know the destination identity. If the destination identity is wrong or has been changed the destination will simply not understand the encrypted e.g. token. With the RSA a-symmetric key cryptography model the authenticity of de client can not be validated without a two way validation model. This means that for real authenticity the server *and* the client must prove their identity to each other e.g. with the Public Key Infrastructure this is done by using the Certificate Authority.

- **Non-repudiation** means that the sender can't deny he send e.g. an encrypted token. As a private key is used to sign and encrypt e.g. a token by a known identity and private key and only a client could know this information so he can't deny only he could have send e.g. the token.

## 2.2 How does the Identity-Based Cryptography model work?

Identity-Based Cryptography is invented by Adi Shamir in 1984[15]. The model is based on the RSA a-symmetric key cryptography model.[18].

The original Identity-Based Cryptography model is based on a trusted third party, the so called *Private Key Generator*. The basic idea behind this model is explained with figure 2.2. Client 1 generates a public key from the master public key, which is public available, and the identity of the destination: client 2. Then client 1 retrieves the private key from the *Private*

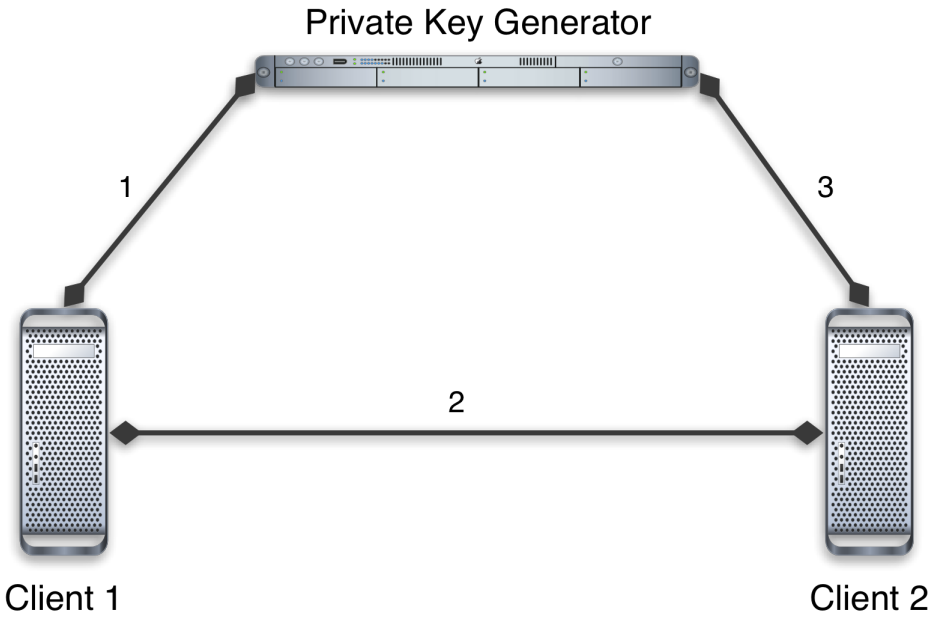


Figure 2.2: The architecture of the original Adi Shamir Identity-Based Cryptography network.

*Key Generator(1)* for the destination identity: client 2. Now with the public key and the private key client 1 encrypts e.g. a token and sends it to client 2 (2). Client 2 also generates the same public key with the identity of client 2. Then client 2 retrieves the private decrypter key from the *Private Key Generator* by the identity of client 1 (3). Client 2 has now enough information to decrypt the encrypted e.g. token from client 1.

This means that the *Private Key Generator* knows the private master key because this trusted third party generates all the keys for decryption and encryption for all the clients. This also means that this trusted third party can decrypt all encrypted e.g. tokens without authorization. This is known as the Key Escrow[17] problem e.g. a commercial company can play the role of the *Private Key Generator* which means that a commercial company can decrypt the communication between e.g. hospitals like patients their medical information. Which ofcourse no patient wants and is even illegal in the Netherlands. But not only the *Private Key Generator* can decrypt all e.g. tokens, but also any identity which plays the role of the server because a retrieved private decrypter key can not only be used for decrypting of e.g. tokens of a specific (server) identity but also the identity of other servers. This means that one server can decrypt e.g. the tokens for another server which are send by the same client if the server knows the identity of the other servers.
Another issue of the original Identity-Based Cryptography model is that it creates the same issue as it wants to resolve. Because all encrypting and decrypting keys have to be send between the *Private Key Generator* and the clients over an open network. This means that every one who can see the network traffic can steal any private key and public keys. The network **is** open because if the network was secure there was no need for encryption. Also the orginal Identity-Based Cryptography model has like the Public Key Infrastructure a serious scalability problem. Since all the client share the same *Private Key Generator*. This means their is a maximum number of clients that could be served concurrently by the *Private Key*

*Generator*. Also if the private master key is stolen every client should update their keys which ofcourse is an almost impossible task. Because of these issues several other systems are proposed that aren't violating the original Identity-Based Cryptography model:

- **Secure Key Issuing Cryptography**[15] is a cryptography model, which distributes parts of the private (master) key over several different trusted third parties so that no trusted third party on his self can decrypt any e.g. token. The destination server can decrypt e.g. the token after contacting every trusted third party. This does not solve the issue of sending keys over an open network and the scalability issue. But the model does solve the Key Escrow issue since no single trusted third party can decrypt e.g. a token.

- **Certificateless Cryptography**[15] is a cryptography model, which shares the generation of the public key and the private key between the destination server and the trusted third party. This does solve the issue that keys are send over an open network because the key that is send over the open network is only half of the public key and private key. Also this solve the Key Escrow issue because the third party has not enough information to decrypt any e.g. token. But this cryptography model stil has a scalability issue because the model is still using a server shared trusted third party.

- **Certificate-Based Encryption**[11] is a cryptography model, which is used by the modern Identity-Based Cryptography solutions because it solves all previous issues. This cryptography model doesn't need a trusted third party. But it does not violate the original Identity-Based Cryptography model of Adi Shamir because the client and the server shares the role of the trusted third party. With this cryptography model the user doubly encrypt and send e.g. a token by using the public key of the destination based on the identity of the destination and a private key. Because the client does not know if the server can decrypt the send e.g. token the model is based on the fact that only the right server can decrypt his own e.g. token. Which means that to make this cryptography model work both client and server should have the same private master key, public key and should know each other their identity.

Because the Certificate-Based Encryption model is the only Identity-Based Cryptography model which can be considered secure in an open network environment, the report is focussed on the Certificate-Based Encryption model.

The Certificate-Based Encryption model uses a shared private master key and public key between the server and the client to encrypt and decrypt e.g. tokens. Also the client and the server should have an identity which can be a user, service or server identity. Which is needed to makes the encrypted e.g. token unique for the identity of the destination.

Before the Identity-Based Cryptography model works, the identities in the network need an initial setup as with the Public Key Infrastructure model, but now also the clients need an initial setup which gives the Identity-Based Cryptography model the possibility to encrypt and sign e.g. the tokens without any cryptography initialisation between the client and the server. When the network is setup, the identities of the users, services **or** servers are distributed. Also for each relationship between two identities, like client and server(group), a private master key and public certificate is generated and distributed to both identities. Because the private master key and the public key are for a human impossible to remember it should be distributed on more fashionable way e.g. by a secure mail or physically on a

secure medium. But sometimes this could be better done by a key distribution server e.g. for a group of replicated servers or a group of clients in a highly trusted network like a computer cluster so each client could use cryptography without each client have to be configured.

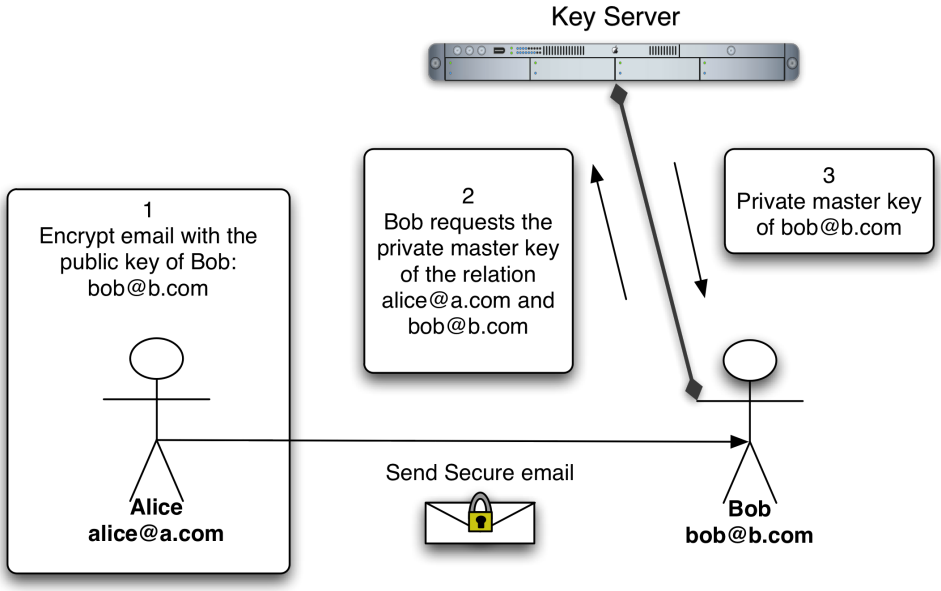


Figure 2.3: An example of sending a secure mail by the Identity-Based Cryptography model by using an user-based identity.

Figure 2.3 is an example of an Certificate-Based Encryption system which send secure mails between users Alice and Bob (The numbers in the figure are relative to the numbers in the enumeration):

1. Alice wants to send a secure email to Bob. This means **Alice knows Bob and his identity bob@b.com, the private master key and the public key she shares with bob@b.com**. Now Alice combines the public key with identity of bob@b.com. Then Alice generates from her private master key a private key, which is with the identity based public key used for encryption. But also the secure mail is signed, by hashing the encrypted e.g. token and also encrypted with the same private key and identity based public key. When the mail is encrypted and signed by Alice the mail is send to Bob.

2. Bob recieves the mail and sees it comes from Alice her identity, alice@a.com. So Bob request the private master key and public key for the relation between alice@a.com and his identity, bob@b.com, from his (shared) key server. This means **the key server knows the relationship between the identity of Bob and Alice and their shared private master key and public certificate.**

3. After Bob knows the private master key Alice used. Bob validates the integrity of the mail by recalculating the hash, with a hash algorithm[14], of the encrypted mail and comparing it with the decrypted hash equipped with the encrypted mail. If they are the

same the integrity of the encrypted mail is still valid and is it safe for Bob to decrypt the email with the private master key and the right private decrypter key.

## 2.3 The difference between Identity-Based Cryptography and Public Key Infrastructure

With the section "How does the RSA a-symmetric key cryptography model work?" is the Public Key Infrastructure model explained as an example. This section is devoted to define the difference in functionality and implementation of the Certificate-Based encryption model which is based on the Identity-Based Cryptography model and the Public Key Infrastructure model. Both cryptography models uses the RSA a-symmetric key cryptography model but differ in the way the private master keys and the public certificate or key are generated and implemented.

With Public Key Infrastructure the private master key is only known by the server and is used to generate private keys for encryption which are dispatched to the clients by the use of the Diffie-Helman(-Merkle) algorithm. The public certificate is only generated once and is generated out of a complex identity (the name of the person who request the signing, organization name, postmaster, etc.) of the server and out of a description for what the certificate may be used (may it be used for signing, a date until the certificate is valid, etc.)[20]. The public certificate should, if it is correct implemented, signed by a trusted third party also called a Certificate Authority. The private key in combination with the signed public certificate is used by the client to encrypt and sign e.g. a token which could only be decrypted by the server. So with the Public Key Infrastructure model the public certificate is the base of the cryptography model, it is what the client needs for confirming the identity of a server.

With the Identity-Based Cryptography model the base of the cryptography model isn't only the public key but also the private master key and the identity of the destination, which all are shared between the server and the client. Because of these values this Identity-Based Cryptography model works without a cryptography initialisation fase between the client and the server or a trusted third party which is required for the Public Key Infrastructure model. This is possible for this Identity-Based Cryptography model because the private key, which is generated from the private master key and the public key combined with the destination identity can be generated by the client. This is everything that is needed to encrypt e.g. a token and hash e.g. a token to ensure the integrity of an encrypted token.

So what does this say about Identity-Based Cryptography and Public Key Infrastructure:

- The Public Key Infrastructure model is based on public values or values that are publicly available to encrypt e.g. tokens. This makes it possible for any client to joins the model. The Identity-Based Cryptography model is based on only private values to encrypt e.g. tokens. This means the user should have information over the identities in the netwerk and should already be known by the netwerk. This concludes that Public Key Infrastructure model can work with anonymous clients but the Identity-Based Cryptography model can **not**.

- The Public Key Infrastructure model needs a Certificate Authority to prove the integrity of the public certificate so the client knows that the certificate is still valid but this also gives a scalability problem which the Identity-Based Cryptography model

solves. Imagine that millions of clients want to access different servers with the Public Key Infrastructure model with a single trusted Certificate Authority. This means that millions of clients validating hundreds of servers their public certificates on a single highly trusted Certificate Authority system is obviously an impossible task. So the role of Certificate Authority is distributed over several systems with many duplicated servers which have very tight policies on secrecy of the private master keys. So this is a structural problem of the Public Key Infrastructure model. The Identity-Based Cryptography model solves this problem by generating any needed values for encryption on the client. This means that these values are instantly trusted like the identity based public key and that's why there is no need for a Certificate Authority which validates these values. Which raises an issue for the Identity-Based Cryptography model: How does the Identity-Based Cryptography model distribute the private master keys, the public keys and identities to generate these public keys? It doesn't the Identity-Based Cryptography model expects from a network that every server and client has his own identity and shared private master key and public certificate between the server and the client.

- The Public Key Infrastructure model can simply invalidate a certificate by the use of an invalidation list on the Certificate Authority. So when the client tries to validate an invalidated certificate, the Certificate Authority invalidate the certificate and the client can search for a new version of the public certificate. With the Identity-Based Cryptography model the shared private master key and public key can be invalidated by a request of the client, which then can update every server in the network which has the identity of the client, but if several servers has the same private master key and public certificate of the client it is a hard task to synchronize all the servers and would maybe need an aggressive synchronization protocol. To let the client transparent connect to every part of the network on the same way. The synchronizing of all servers could be accelerated by the use of a highly trusted shared key server. If the server invalidates the private master key and public key of the user. This means the client can't login anymore and should get a new private master key and public key for his identity if you isn't yet already informed.

- In the Public Key Infrastructure model the public certificates are very complex[20] and needs to be signed by a Certificate Authority. This means that a normal client, which most times have a variable address, will almost never want to be validated by a Certificate Authority. So it is technically possible to generate a public certificate for the server as well for the client to prove a two way authenticity but it is not likely to implement such a scheme. With the Identity-Based Cryptography model all the information to proof a two way authenticity is already on the client and on the server. So the server and the client can decrypt and encrypt e.g. tokens because of the shared private master key. So when the server gets an encrypted e.g. token from the client it could, after decryption, send it encrypted back but then with the public key with the identity of the client. If both succeed, the authenticity of both the client and server is proven because both the identities have proven that they know each other identity, public key and a private key.

## 2.4 The difference between Identity-Based Cryptography and Kerberos

Kerberos[16] is a *Single Sign On* system, which means that the clients have to authenticate only once to access several *Service Servers* with specific resources within a specific time frame. Kerberos uses a symmetric key cryptography to negotiate a session key with the use of a *Ticket Granting Server*. This *Ticket Granting Server* authenticates the client by using an *Authentication Server(1)*. This session key, called a *Ticket Granting Ticket*, is used to access several *Service Servers* without direct authentication. The client uses the *Ticket Granting Ticket* to request a session key for a specific service from the *Ticket Granting Service(2)* to access the *Service Server(3)*.

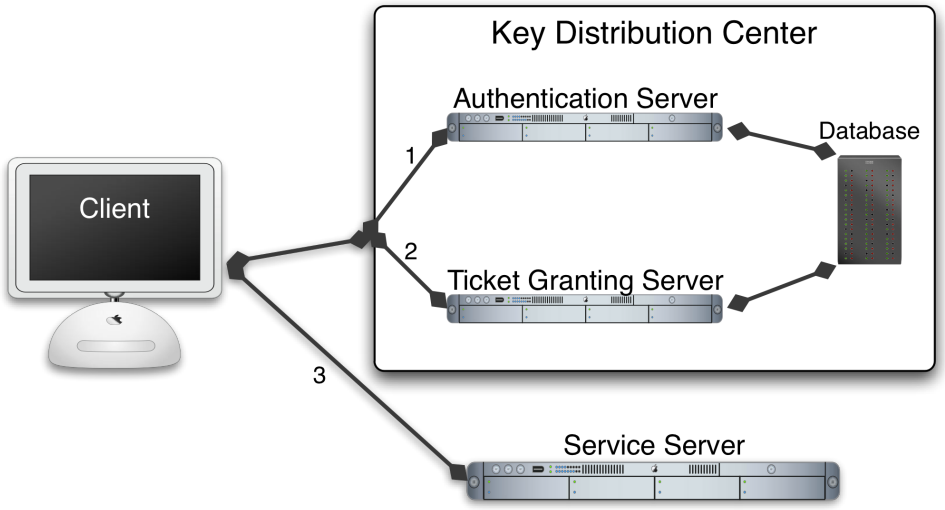


Figure 2.4: An abstract overview of kerberos.

The main differences between Kerberos and Identity-Based Cryptography is that there is no initialization fase between the client and server with Identity-Based Cryptography. Also Kerberos uses symmetric key cryptography which means that the password is send encrypted over the network but because the client does not check the authenticity of the server. This means the client could send his encrypted password to anyone. But also the integrity of each message in the Kerberos system isn't checked which means that any data that is send with the Kerberos system can be manipulated or mutalated without the server or client even knows it. Kerberos is obvious not a real concurrent on cryptography for both RSA a-symmetric key cryptography models. Besides the lack of several security features Kerberos is a more complete system then the RSA a-symmetric key cryptography models because Kerberos has a build-in authentication, accounting and authorization system and the RSA a-symmetric key cryptography model only an authentication model and a small authorization model in the form of a public certificate.

## 2.5 The security of the Identity-Based Cryptography model

In this section the security of the Identity-Based Cryptography model is evaluated. It is important to mention that any of the discussed issues are pure theoretical based on an abstract model, which means that none of them are tested.

### 2.5.1 Identity length

It maybe looks like the length of the identity or the form changes the strength of the public key but the identity is purely used to ensure e.g. the token is encrypted for the right destination. So if the client is communicating with the wrong identity the identity could not decrypt e.g. the token. The identity is mixed in the cryptography by combining it with the public key which is a secure big random number created from two big prime numbers which also creates the private master key.

### 2.5.2 The private (master) key generation

Generating the private (master) key is the weak point of the RSA a-symmetric key cryptography model. The private master key should be a real big number which should be calculated from two **randomly** chosen prime numbers[18]. Also the private key is generated by a **random** number from the private master key. So the weak point is the way a random function work. If a hacker could predict or guess the next random number a hacker could know the next generated two prime numbers which generates the public key and the private master key or the next private key. So these random values must be generated with great care and the random functions have been optimized over the years that are used for creating these big numbers[18].

### 2.5.3 The private key on the Certificate-Based Encryption system

The private key is based on a random number when e.g. a token is encrypted it can't be distributed with the initial setup of the network. But if this is sent decrypted with the encrypted e.g. token it should be safe because from a single random number the data could not be decrypted but ofcourse this should not be an option. The solution which several Identity-Based Cryptography solutions use is using a Pairing Algorithm with a map[9]. The pairing algorithm is implemented with a bilinear map which always return the same result of two linear functions, which needs two numbers. The pairing algorithm ensures:

$$Pair(a \times X, b \times Y) \equiv Pair(b \times X, a \times Y) \tag{2.7}$$

The Certificate-Based Encryption system uses this fact to exchange the private keys. First the client encrypts and signs e.g. a token with a private key which is generated by using the pairing algorithm:

$$privateKey = Pair(randomNumber \times identityDestination, secret \times randomEllipticCurveValue) \tag{2.8}$$

Above there is a elliptic curve value. This value is a random number which points with the private master key to the private key from the RSA algorithm. Described as: A co-prime

with the private master key which has a value between one and the private master key. Now the encrypted and signed e.g. token is send to the destination identity with the product of:

$$additionalSendNumber = randomNumber \times randomEllipticCurveValue \tag{2.9}$$

The destination server needs the privateKey to decrypt e.g. a token. This done by the pairing algorithm:

$$Pair(randomNumber \times identityDestination, secret \times randomEllipticCurveValue) \tag{2.10}$$

**is equivelant to**

$$Pair(secret \times identityDestination, randomNumber \times randomEllipticCurveValue) \tag{2.11}$$

Since we have the $randomNumber \times randomEllipticCurveValue$ answer the new formula is:

$$privateKey = Pair(secret \times identityDestination, additionalSendNumber) \tag{2.12}$$

This can be solved and then the private key is secure transferred from the client to the server.

### 2.5.4 Replay attack

The Identity-Based Cryptography model could be vulnerable for replay attacks. This attack is based on intercepting data that has been send and re-sending the same data later all over again. If a system works right this data will be ignored, but some systems will react on the same data more then once. These are systems that are vulnerable for replay attacks. Maybe an attacker can't decrypt the message but he can mess up the system by resending previous data. The replay attack is possible because the private keys of encrypted tokens maybe used more then once because with the Identity-Based Cryptography model the private key isn't negotiated with the server as with the Public Key Infrastructure model with the Diffie-Helman(-Merkle) algorithm. So sending more then once the encrypted e.g. token by a client is accepted by the server because the client maybe used more then once the same private key. This can be prevented by the so called nonce: Number Only used oNCE. This means that every private key may only be used once within a certain time frame e.g. once per week. But then after a week the attack could be completed. So also the identity must contain the week number of the year. This works because the random number that is chosen to generate these client private keys is so big that it is not likely that the same number is generated more then once but if it happens it could be corrected by a graceful re-send command from the server so the client can re-send e.g. the token with another private key.

### 2.5.5 The server invalidating a stolen private master key

Another issue of the Identity-Based Cryptography model is created when the server is invalidating a private master key because it has been stolen. If the client doesn't know this it could setup a conversation with a fake server who stole the private master key. So the fake server can steal any information what the client sends or at least the first encrypted e.g. token because after communicating with a real server the client will be informed of the invalidation. This problem does not exist with Public Key Infrastructure because the public certificate is first validated or invalidated, which forces the client to search for a valid public certificate before sending an encrypted message.

# Chapter 3

# Implementation

This chapter evaluates the Identity-Based Cryptography solution of Voltage called Identity-Based Encryption and a soution of the National University of Ireland in Maynooth called Eyebee. Also a short experiment is done with the Eyebee solution.

## 3.1 Known Solutions

The known solutions Eyebee and Identity-Based Encryption are compared.

### 3.1.1 Eyebee

Eyebee[8] is a Java[3] based Identity-Based Cryptography library made in 2004 by the "Computer Security and Cryptography" research group of the computer science department of the National University of Ireland which is located in Maynoot. Eyebee is build on the Java Cryptography Extension (JCE) which is part of the standard Java[3] library since version 1.4 is located in the "java.security" namespace. The version of the Eyebee library that is used for this report is version 1.0.38. Eyebee is documented by the Javadoc webpages with the downloadable library and the source code. Also their is a short implementation example on the website of the research group. Besides the code there is no license described on the site.

### 3.1.2 Voltage IBE

The Voltage Identity-Based Cryptography[9] solution is a C library. It is a professional build library which runs on Microsoft Windows and Linux. It has several technical, mathematical and implementation related documentation on the website and it is easy to implement, by several different interfaces and examples. Voltage also has been awarded by the National Institute of Standards and Technology. On the website it is referred to as:

> FIPS Certification: The Toolkit's Cryptographic Module has been awarded FIPS 140-2 Level 1 certification by NIST. FIPS, the Federal Information Processing Standards, are a set of standards created by the National Institute of Standards and Technology to ensure correct implementation and interoperability of cryptographic systems.

But Voltage has a very strict license e.g. the license gives Voltage the right to change the license without notification to their customers also their product may only be used by universities in a test environment and may not be used by more than 50 clients, which may not be concurrent, by the test environment. The license makes the Voltage Identity-Based Cryptography solution unsuitable for the TBS-TVS project.

### 3.1.3 Comparing Voltage IBE and Eyebee

In this section Voltage IBE and Eyebee are compared. I described in the previous section that Voltage IBE have a license that is not compatible with the TBS-TVS project. But because Voltage have such a well developed solution it isn't a bad idea to still compare it.

| **Compare** | **Voltage** IBE | **Eyebee** |
|---|---|---|
| Hash Algorithms for signing | SHA-1 | MD2, MD5 or SHA-1 |
| Language | C and C++ | Java |
| Supported Operating Systems | Windows and Linux | Every OS that supports Java 1.4 or higher. |
| RSA Library | OpenSSL | Java Cryptography Extension (JCE) |
| Identity-Based Cryptography implementation | Certificate-Based Encryption | Certificate-Based Encryption |

## 3.2 Test Installation

As this report is a sub project of the TBS-TVS project and researches the Identity-Based Cryptography model, a Java[3] class has been written which implements the Eyebee solution.

### 3.2.1 The token handeling and distribution model of the TBS-TVS project using a shared secret

In this section is explained how currently the DES symmetric key cryptography in the TBS-TVS project is implemented.

The TBS-TVS project which is a side project of the TBS-TVS project for developing the *Token Validation Service* is devided in five namespaces within the Java[3] "org.aaaarch" namespace:

- config. This namespace has all the classes used for the key store, which is a local key server to hold all token keys.
- crypto. This namespace has classes that encrypt and decrypt data with the DES symmetric key algorithm.
- signature. This namespace has classes that sign or check the signature of a XML[10] document.
- utils. This namespace has all kind of tools wrapped in classes like converters for hex, XML, datetime formats and IO methods for writing and reading files.

- gaaapi. This namespace exist out three namespaces:

  - common. This namespace exists out of some static variables for the token system and a standard id (number) generator.
  - ticktok. This namespace is for creating, editing and reading tokens.
  - tvs. This is the namespace for the Ticket Validation Service. This namespace uses mostly the ticktock namespace to validate and manage XML[10] and binary tokens.

Next the TBS-TVS project is further explained and the position of the *Ticket Validation Service* within this system.

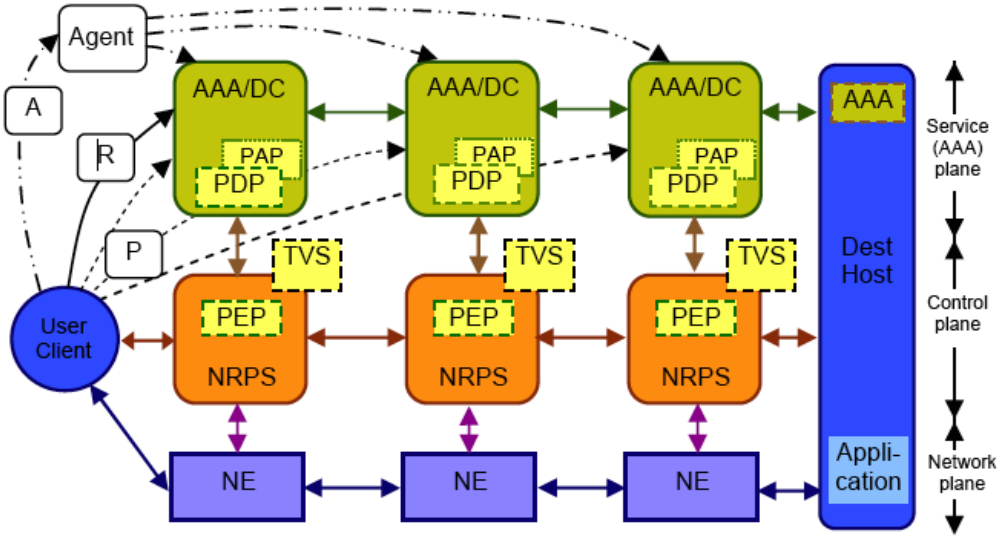


Figure 3.1: Components involved in complex resource provisioning and basic sequences (agent, relay, and polling) of the TBS-TVS system

In figure 3.1 the abstract setup of component of the TBS-TVS project is shown. On the left side an User Client wants to access the network to access a resource on the system at the right side of the figure. To get to the right side the user needs to go thru one of the three layers in the horizontal and diagonal direction. Horizontal there are three similar layers which are three example domains within the TBS-TVS system with each:

- **The AAA layer** is the Authentication, Authorization, Accounting layer. This layer knows which users should have access to the domain (Accounting), how these users must prove who they are to the domain (Authentication) and what rights they have if they prove their authenticity to the domain (Authorization). In the figure it is also defined as DC which means a Domain Controller as implemented by the different operating systems. The TBS-TVS system is implementing these models by two model components:

  - **PAP** means Policy Authority Point. This is the component which decides which users may login (Accounting) and how they must authenticate(Authentication).

  - **PDP** means Policy Decision Point. This is the component which decides which rights the client has, which are defined by several policies (Authorization).

- **The NRPS layer** is the Network Resource Provisioning System which gives a token access to a resource or routes the token to a NPRS who can gives the token access to the resource. The following components provide these services:

  - **PEP** means Policy Enforcement Point which is a component that enforces a new policy for a domain or a domain the token is forwarded to. This means that the authorization of the token changes.
  - **TVS** means Token Validation Service which is a component which validates a token and gives a resource to the token or routes the token to a domain which has the right resource.

- **The NE layer** exist out of Network Elements of the domain. This component could be seen as the network infrastructure.

The Destination Host has three layers but should only be accessed by the middle layer. The client shouldn't get access to the Destination Host by the top layer because this means a client logins on the Destination Host without being authenticated and authorizated by the TBS-TVS system. The client also shouldn't get access to the Destination Host by the bottom layer because this means a client is on the same network as Destination Host and gets direct access to a service of the host with only a partial view of the entire system e.g. a resource could be disconnected of the TBS-TVS system or the TBS-TVS system has a task for the Destination Host with a higher priority.

The DES symmetric key cryptography[12] is implemented in the TVS library (org.aaaarch.gaaapi.tvs) by the main "TVS" class. This uses a "TokenBuilder" class to decrypt tokens by using the "TokenKey" class which uses a DES algorithm (org.aaaarch.crypto) and SHA-1 hashing algorithm[14] (org.aaaarch.gaaapi.HMACprocessor) wrapper classes. The *Token Validation Service* needs to decrypt the tokens to reservate the right resource or routes the token to the right domain which has the right resource.

### 3.2.2 The token handeling and distribution model of the TBS-TVS project using an Identity-Based Cryptography

In this section is explained how the Eyebee, Identity-Based Cryptography, solution works and how this replaces the current symmetric key cryptography. The org.aaaarch.ibc namespace is created for these new Eyebee Identity-Based Cryptography test and IBC class. The IBC class have 6 methods:

- **IBC.** This is an initialization method. Without an argument the default SHA-1 hash algorithm[14] is used else MD2 or MD5 can be chosen by a first "String" argument.

- **GenerateMasterKey.** This method is used to generate the private master key.

- **EncryptToken.** This method is used to encrypt a token or any other byte array by 3 arguments: the private master key as BigInteger, the destination identity as String and a Byte array for the data to encrypt.

- **DecryptToken.** This method is for decrypting a token or any other byte array by 3 arguments. the private master key as BigInteger, the destination identity as String and a Byte array for the data to encrypt.
- **ToXML.** This method is for the TBS-TVS system. Since the Eyebee solution uses a Java[3] object around the encrypted data and pairing algorithm variable. First these variables needs to be extracted and then put in a XML structure to fit to the requirements of the TBS-TVS project.
- **ToEncryptedToken.** This method is for putting the XML data back into the Java[3] Eyebee Object so the Eyebee library can decrypt the data.

Next the process of Identity-Based Cryptography is explained. But now with the Eyebee implementation besides it. This shows how Eyebee implement Certificate-Based Encryption, which is an implementation of Identity-Based Cryptography.

1. **Setup the Eyebee Identity-Based Cryptography network.** Before the client can send encrypted e.g. tokens to the server the network needs to be setup. This means for the Identity-Based Cryptography model that a private master key and identity is created and distributed between the client and the server.

   (a) **Generating the private master key.** Eyebee uses for generating the private key a so called map, which is also used indirect for the private master keys. The private master key gives the starting position in the map.

   > BigInteger masterKey = new BigInteger( map.getQ().bitLength() - 1, new SecureRandom() );

   This code creates a big number, in Java so called BigInteger, from a random randomly chosen row.

   (b) **Distribute the private master key between the client and the server** The client and the server needs this private master key to decrypt and encrypt e.g. tokens which are send to each other. So the client or server will now distribute this key after it is generated to the other party. This isn't part of the Identity-Based Cryptography model and also not a part of Eyebee solution but for now lets imagine it has been send in an encrypted file by mail.

2. **The client wants to encrypt a token and send it to the server.** The clients wants to encrypt a token. This means the client has to generate a public key and a private key to encrypt and sign the token. For creating the keys a pairing algorithm is used with a bilinear map as discussed in the section "The private key on the Certificate-Based Encryption system".

   (a) **Generating the public key.** The public key is generated from the private master key by the use of the map and the identity of the server. Since the map in reality is a complex mathematical function which goes beyond the scope of this document it is enough to say that the map can also generate the public key by the private master key. To generate a complete public key it is combined with the identity of the destination. Since the identity of the destination is known by the client he could input this e.g. "token-server@uva.nl".

IbeSystemParameters systemParameters = new IbeSystemParameters( map, hash, masterKey );
IbeKeyParameters keyParameters = new IbeKeyParameters( hash, identity );
PublicKey publicKey = new IbePublicKey( keyParameters.getPublicKey() );
cipher.init( Cipher.ENCRYPT_MODE, publicKey, systemParameters, new SecureRandom() );

First the systemParameters are initialised: the map from where every key is generated, the hash function which is needed to sign an encrypted token and the masterKey which gives the right row of private keys. Next the keyParameters is given which is the identity of the destination which is hashed for making it more variable then only keyboard-based characters and make it unreadable and unresolvable for people on the network who should not know to easy for which identity a token is encrypted. Next the publicKey is generated: the hashed identity. On the last line the cipher is initialized. The publicKey, map and private masterKey are used to create the final public key based on the identity of the destination.

(b) **Generating the private key.** Before the token could be encrypted a private key is needed. This generated by choosing one from the map of the row of the master key. So a random number combined with the master key points to a private key in the map.

IbeSystemParameters systemParameters = new IbeSystemParameters( map, hash, masterKey );
cipher.init( Cipher.ENCRYPT_MODE, publicKey, systemParameters, new SecureRandom() );

Selecting a private key in the map is done by Eyebee by the systemParameters which has the map and private masterKey. The systemParameters is called by the initialization of the cipher with a secure random number.

(c) **Encrypt the token** Now the client can securely send the server the token after encrypting it.

cipher.init( Cipher.ENCRYPT_MODE, publicKey, systemParameters, new SecureRandom() );
encryptedToken = cipher.doFinal( token );

The function "dofinal" seams strange but since Eyebee is build on a Java based cryptography frameworks which supports algorithms which supports multiply blocks it is needed to inherent these functions. Since Identity-Based Cryptography doesn't has multiply block support it is created with a single encryption or decryption step which uses the mathematical procedure with the private key and public key described by the section "How does the RSA a-symmetric key cryptography model work?". The result is a byte array which is safe to send over the network to the server.

3. **The server recieved the encrypted token and decrypts it.**

(a) **Generating the public key.** The server generates the public key by the destination identity, the private master key and the map as done by the client. The

destination identity is still the same identity: the server his own identity.

```
IbeSystemParameters systemParameters = new IbeSystemParameters( map,
hash, masterKey );
IbeKeyParameters keyParameters = new IbeKeyParameters( hash, iden-
tity, masterKey, map );
kpg.initialize( keyParameters );
KeyPair keyPair = kpg.generateKeyPair();
PrivateKey privateKey = keyPair.getPrivate();
```

The public key is generated from the private masterkey, map and the identity of the destination. The masterkey and map is retrieved from the systemParameters. The identity is retrieved from the private key from the pairing function. The private key of the pairing function is based on the product of the private master key and the identity the identity can be resolved since the private master key is known. Now the public key can be created.

(b) **Generating the private key.** The private key which the client used can be resolved by using the pairing algorithm with the encrypted token. To retrieve the private key from the encrypted token the identity of the destination and the shared secret is used. The shared secret is the private master key because this is the only private value which is shared by the client and the server.

```
IbeKeyParameters keyParameters = new IbeKeyParameters( hash, iden-
tity, masterKey, map );
kpg.initialize( keyParameters );
KeyPair keyPair = kpg.generateKeyPair();
PrivateKey privateKey = keyPair.getPrivate();
```

The private key is resolved by the pairing key function from the encrypted token. On the second and third line a kpg (Key Pair Generator) creates the keypair of the product of the private masterkey and the identity of the server which can be both found in the keyParameters and the number that has been send with the encrypted token. With this keypair the private key is calculated by the use of the map.

(c) **Decrypting the token.** Now the private key is known which the client used and the private decryption key can be calculated. With this private decrypter key and the public key the server can decrypt the token.

```
cipher.init( Cipher.DECRYPT_MODE, privateKey, systemParameters );
token = cipher.doFinal( encryptedToken );
```

The cipher uses the systemParameters and pairing function to generate the public key. Also the cipher uses the privateKey and the private master key from the systemParameters to generate a private decryption key. Then the private decryption key and public key are used by the cipher "doFinal" method to decrypt the token and return it to the server.

# Chapter 4

# Experiment and Evaluation

Now the Eyebee solution is explained a short experiment is held to see if the output of the Eyebee solution is fits the code. How it is explained.

## 4.1 Experiment description

A test class is created to see if the output created by Eyebee fits the code. The test class exists out of a simple sequence of generating the private master key, encrypting the text "Test Token Key #1" to identity "Rudy.Borgstede@gmail.com" and the private master key and then the encrypted message is again decrypted with the identity and the private master key. The most important information is extracted on different points of the process: The input string, the identity, the private master key, the public key and the private key.

### 4.1.1 Output

## 4.2 Evaluation of the experiment

As evaluation of the experiment the output of the values of the extracted information are shown and explained. Also a short section is devoted to the performance of the test program.

### 4.2.1 Output

For this report the ouput is structured and only shown once per item since e.g. the private master key did not change for encryption and decryption.

- **Input:** Test Token key #1
- **Identity:** Rudy.Borgstede@gmail.com and hashed with SHA-1:

  95 6d 74 25 69 46 a5 d0
  81 14 75 e3 f9 4f 0e 83

- **Private Master Key:**

  7c 01 fc 3e 86 c6 cf 51
  60 c5 d5 95 52 1a c4 5f
  c1 5e 7d bb 5e 06 6d 19

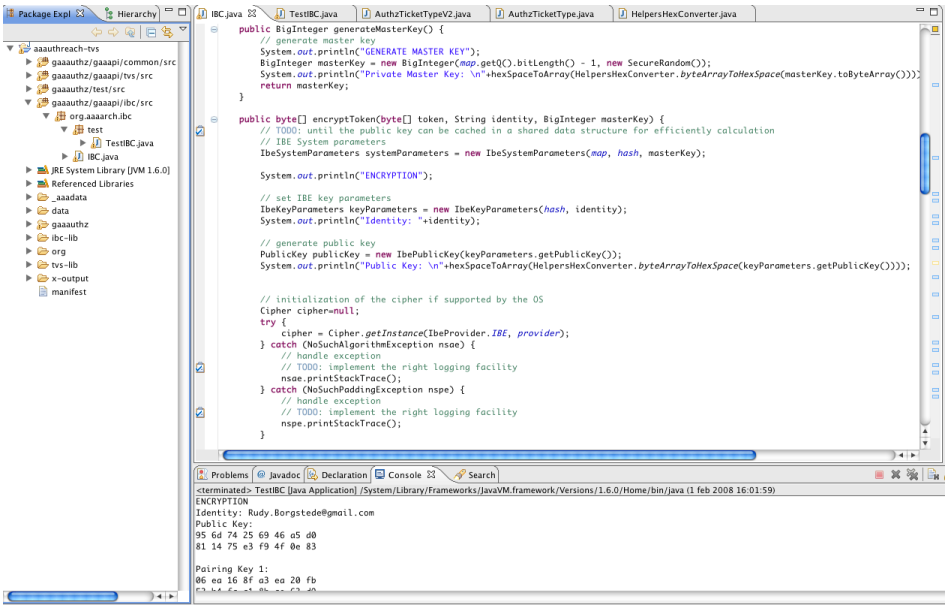


Figure 4.1: A screenshot of the IBM Eclipse Java Development Environment with the test output.

- **Public Key with identity:**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 03 | 26 | 0e | 4b | 97 | 9a | cb | dd |
| b7 | 9a | 57 | b7 | 29 | 3b | cb | 26 |
| 69 | 9e | c9 | 75 | 55 | 9b | e7 | 45 |
| f9 | 7a | f1 | d1 | cb | 8c | 04 | 1e |
| cb | 13 | 9e | 7e | 38 | 99 | 8b | 27 |
| 16 | c3 | a4 | 8f | e6 | 89 | bb | ae |
| 52 | f9 | 1f | a1 | 29 | bc | 20 | 9b |
| 49 | 31 | da | b8 | 91 | a7 | 8e | 4c |

- **Private Key:**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 02 | a7 | 86 | 92 | 99 | d3 | 61 | 64 |
| bc | f7 | 17 | 4c | 32 | 14 | 64 | c1 |
| 4c | 50 | ee | 8c | 72 | 2f | 1b | 07 |
| f5 | 5f | 9c | 10 | 79 | 5f | 82 | 6f |
| 46 | 45 | 1e | cf | 53 | cc | ef | 51 |
| f6 | 25 | 58 | 19 | 90 | ae | 57 | 1f |
| fc | 87 | 65 | cf | ec | 81 | 40 | db |
| 24 | ce | 3b | e8 | a0 | 7c | 39 | a7 |

### 4.2.2 Performance

As the Identity-Based Cryptography Eyebee solution is writen in Java[3] the performance isn't really good. It takes over a second to encrypt and decrypt a test message with a modern macintosh which should be considered to slow for a production environment.

# Chapter 5

# Conclusion

In this chapter an opinion is given about the usefullness of the Identity-Based Cryptography model and the usefullness of the cryptography model for the TBS-TVS project.

## 5.1 Is Identity-Based Cryptography usefull?

The modern Identity-Based Cryptography model like Certificate-Based Encryption is very usefull because it gives the client the possibility to send encrypted information without showing almost any information about the cryptography process to the network. The Identity-Based Cryptography model doesn't replaces the Public Key Cryptography model but rather completes it because the Public Key Cryptography model only (without dismissing private keys) supports anonymous clients while the Identity-Based Cryptography model only supports non-anonymous clients. The use of these two models gives a client the possibility to complete secure communicate between the client and the server on a personalized way, because anonymous is also an identity.

## 5.2 Is Identity-Based Cryptography usefull for the TBS-TVS project?

The Identity-Based Cryptography model is a good solution for the TBS-TVS project because it gives a high level of security from a fast setup that does not voilate the boundries of the Local Area Network. This means for the TBS-TVS project that resources and domains could be reasonably fast connected to the network and disconnected on a secure way and that domains could be setup without extensive and expensive configurations like with the Public Key Infrastructure model. The Eyebee solution that has been tested could not be used for TBS-TVS project because it hasn't got enough performance to work in a production environment. So other Identity-Based Cryptography solutions must be researched which have a high performance library.

# Chapter 6

# Further research

In this chapter subject are discussed which should be further researched.

## 6.1 Standarization

The Identity-Based Cryptography model is not yet a standard, even not the model of Adi Shamir, because it is implemented differently by every Identity-Based Cryptography system. The Identity-Based Cryptography model just isn't mature enough to be used in a production environment. It needs to be further researched when it is mature enough to use in a production environment.

## 6.2 Scalability

The Identity-Based Cryptography model works in a test environment and small systems but it is not tested in a big environment. So the Identity-Based Cryptography model must be tested for scalability in different challenging environments.

### 6.2.1 Key Server

A scalability issue that needs to be researched is the distribution of the keys and the identities on the network. But also the availability of the keys for any identity on the network because for big (server) systems the keys should be shared by a secure network database, but normal clients will only want a small and fast local key store. This isn't defined by the cryptography model.

### 6.2.2 Performance

The performance of the Identity-Based Cryptography model must be extensively researched and tested. Performance in the sence of the number of clients that can be served by a single Ghz and how stable the model is. This is needed for implementing the model on systems with a big number of participents and for creating a business model for the Identity-Based Cryptography model.

## 6.3 Security

The security of the Identity-Based Cryptography model should not be a problem because when the cryptography of the model is still based on the RSA a-symmetric key cryptography it could use the vurnabilities fixes of other RSA a-symmetric key cryptography models and systems e.g. the mature Public Key Infrastructure model. Besides looking at other systemsm and models the security of the cryptography model should still be tested and evaluated.

## 6.4 P2P

Routing the tokens within a domain is hard to do without decrypting them. Which means that optimizing the route is really optimizing the TBS-TVS system e.g. choosing identities which says something about the address could give a domain hints to optimize the token. To maybe further optimize this routing of tokens P2P(Peer to Peer) could be used, since the network infrastructure should not change much. This means that routes that are used much could be autonomous be highly optimized.

# Chapter 7

# Appendix

- Bibliography
- Planning
- Log
- Project description

# Bibliography


[1] C. de Laat. Research project 1 presentation, januari 2008. Available from: http://staff.science.uva.nl/~delaat/talks/CdL-2007-10-02.pdf [cited 26 januari 2008].

[2] Y. Demchenko. Aaauthreach, januari 2008. Available from: http://uazone.org/demch/projects/aaauthreach/index.html [cited 16 januari 2008].

[3] sun Microsystem. Sun java, januari 2008. Available from: http://java.sun.com/ [cited 16 januari 2008].

[4] Surfnet. Public key infrastructure, januari 2008. Available from: http://aaa.surfnet.nl/info/pki/home.jsp [cited 16 januari 2008].

[5] university of Amsterdam. Network description language, januari 2008. Available from: http://www.science.uva.nl/research/sne/ndl [cited 16 januari 2008].

[6] university of Amsterdam. Os3, januari 2008. Available from: http://www.os3.nl [cited 24 januari 2008].

[7] university of Amsterdam. Universiteit of amsterdam, januari 2008. Available from: http://www.uva.nl [cited 24 januari 2008].

[8] university of Ireland. Eyebee identity-based cryptography, januari 2008. Available from: http://www.crypto.cs.nuim.ie/software/eyebee/ [cited 16 januari 2008].

[9] Voltage. Identity-based encryption, januari 2008. Available from: http://www.voltage.com/ibe_dev/ [cited 16 januari 2008].

[10] Wikipedia. Xml. Available from: http://en.wikipedia.org/wiki/Xml [cited 24 januari 2008].

[11] Wikipedia. Certificate-based encryption, januari 2008. Available from: http://en.wikipedia.org/wiki/Certificate-based_encryption [cited 27 januari 2008].

[12] Wikipedia. Des, januari 2008. Available from: http://en.wikipedia.org/wiki/Data_Encryption_Standard [cited 16 januari 2008].

[13] Wikipedia. Euler's totient, januari 2008. Available from: http://en.wikipedia.org/wiki/Totient [cited 24 januari 2008].

[14] Wikipedia. Hash, januari 2008. Available from: http://en.wikipedia.org/wiki/Cryptographic_hash_function [cited 25 januari 2008].

[15] Wikipedia. Identity-based cryptography, januari 2008. Available from: http://en.wikipedia.org/wiki/Identity_based_encryption [cited 27 januari 2008].

[16] Wikipedia. Kerberos, januari 2008. Available from: http://en.wikipedia.org/wiki/Kerberos_(protocol) [cited 16 januari 2008].

[17] Wikipedia. Key escrow, januari 2008. Available from: http://en.wikipedia.org/wiki/Key_escrow [cited 27 januari 2008].

[18] Wikipedia. Rsa, januari 2008. Available from: http://en.wikipedia.org/wiki/Rsa [cited 24 januari 2008].

[19] Wikipedia. Security token, januari 2008. Available from: http://en.wikipedia.org/wiki/Security_token [cited 16 januari 2008].

[20] Wikipedia. X.509, januari 2008. Available from: http://en.wikipedia.org/wiki/X.509 [cited 26 januari 2008].

# Planning

### Week 0 - Before the start of the project

- Make the project description
- Study the Java TBS-TVS project
- Learn more about Identity-Based Cryptography
- Find Java Identity-Based Cryptography implementations

### Week 1 - 7 January 2008

- Make a test implementation of the found Identity-Based Cryptography solution for the TBS-TVS project
- Finalize the planning and mail it to Yuri Demchenko and Cees de Laat on 11 January 2008

### Week 2 - 14 January 2008

- Evaluate the Identity-Based Cryptography implementation
- Identity-Based Cryptography experiment on the TBS-TVS project

### Week 3 - 21 January 2008

- Evaluate experiment
- Make report

### Week 4 - 28 January 2008

- Evaluation report
- Finishing report before 4 February and sent to Cees de Laat and Yuri Demchenko
- Make presentation and evaluate

### Week 5 - 4 February 2008

- Presentation on the $6^{th}$ of February 2008[1]

[1] At the CWI on the kruislaan in the auditorium Z009

## Log

| | |
|---|---|
| 5 december 2007 | First appointment with Yuri Demchenko at the science park of Amsterdam |
| 25/26 december 2007 | Reading Identity-Based Cryptography information that I got from Yuri Demchenko. |
| 27 december 2007 | Make a appointment with Yuri Demchenko by mail. Getting feedback on some questions about the TBS-TVS project and the concept of Identity-Based Cryptography |
| 2 January 2008 | Studied the websites that Yuri Demchenko send me by mail. |
| 3 January 2008 | Appointment Yuri Demchenko at the science park of Amsterdam. Discussing direction of the project: Identity-Based Cryptography. |
| 5 January 2008 | Start of the report by documenting the previous discussed planning and making a basic setup for the report. |
| 9 January 2008 | Reading the Identity-Based Cryptography documentation given by Yuri and finding Java Identity-Based Cryptography implementations and reading their information. |
| 10 January 2008 | Creating Java code for the Identity-Based Cryptography implementation. Based on the eyebee project. |
| 11 January 2008 | Creating testcode for the made Identity-Based Cryptography Java class. Optimizing the made Java class of the eyebee implementation. Send Cees de Laat a copy of my planning. |
| 13 January 2008 | Mailed Yuri Demchenko for a new appointment. |
| 14 January 2008 | Updating the log. |
| 15 January 2008 | Appointment with Yuri Demchenko. Updating the report structure. |
| 16 January 2008 | Creating chapters 1 and 2.1 of the report. |
| 17 January 2008 | Evaluated the report and finished chapters 1 until 2.3 of the report. |
| 18 January 2008 | Writing chapters 1 and 2. Email Yuri Demchenko the status of the report. |
| 19 January 2008 | A fast evaluation of the report. |
| 20 January 2008 | Making a new appointment with Yuri Demchenko. Checking the spelling and the correctness of the report. Updating the log. |
| 21 January 2008 | Checking the spelling of the report by a third person. Report version 0.1. |
| 22 January 2008 | Appointment with Yuri Demchenko. Got some changes for the report. |
| 23 and 24 January 2008 | Creating chapter 3 and moving the Identity Based-Cryptography code to the right namespace. |
| 25 January 2008 | Checking the logic and spelling of the report. Finish report until chapter 2. |
| 26 January 2008 | Finished report until chapter 3.2 |
| 27 and 28 January 2008 | Created a draft of the rapport except for chapter 4. Report version 0.2. |
| 29 January 2008 | Appointment Yuri Demchenko. |
| 30 January 2008 | Wrote a section about the Pairing Algorithm and did a final spelling check. Report version 0.3. |

| | |
|---|---|
| 31 January 2008 | Final appointment with Yuri Demchenko. |
| 1 February 2008 | Created final draft of the total report version 0.4. |
| 4 February 2008 | Fix the last faults. By the comments of Yuri Demchenko |

## Project description

Project as described on the original project site[2]:

> **9 Security model for hybrid token-based networking models.**
> The Token Based Networking (TBN) is based on using security token (both binary and XML) for controlling user access to the reserved network resources. The TBN allows separating the reservation and authorisation stage (that is typically complex and slow) and access or consumption stage. The project will research different token based access control models for hybrid/combined GMPLS and RSVP based networks that may use binary or XML tokens. The project will investigate two basic models for token and token key generation: with the shared secret (SC) and using Identity-Based Cryptography (Identity-Based Cryptography). Additionally, the project will look at the security issues with the generation and distribution of tokens and token keys. Proposed solution will be modeled as a special Token Validation Service (TVS) that is considered as a pluggable component to major Network Resource Provisioning Systems (NRPS) to enable TBN functionality.
> *Yuri Demchenko*

[2] http://staff.science.uva.nl/~delaat/sne-2007-2008/index.html